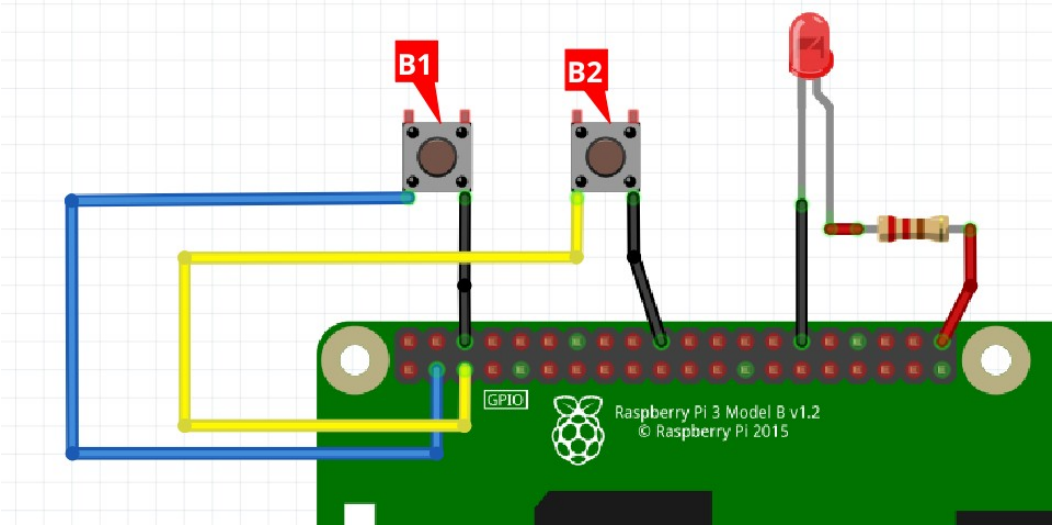
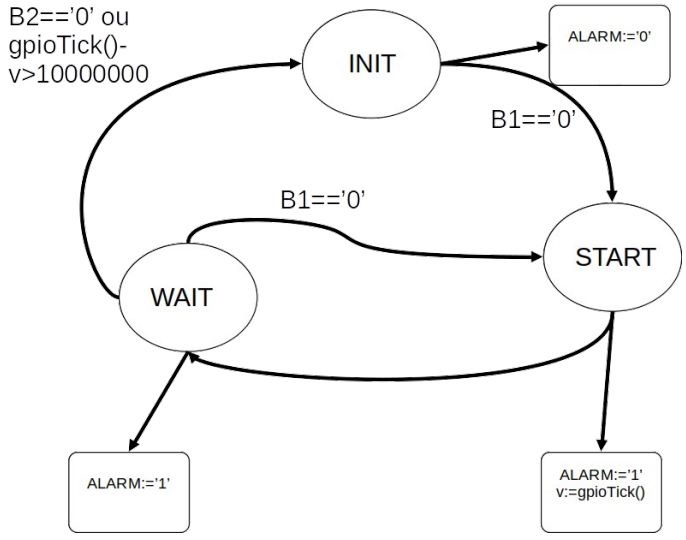
Université Tunis El-Manar	**Année Universitaire** : 2022-2023
Faculté des Sciences de Tunis	**Module :** Conception des objets connectés
Section : LIOT3	**Enseignant :** C.A. ABID

Exercice 1. (correction)
On se propose de simuler un système d'alarme dont le montage est présenté comme suit :



L'alarme se déclenche lorsque le bouton poussoir B1 est appuyé. Cela se fait en flashant la diode LED en sortie pendant 10 secondes. Si le bouton B1 est appuyé encore une fois, la durée de l'alarme est ré-initialisée. Par contre, si le bouton B2 est appuyé, l'alarme s'arrête immédiatement.

1) Donner la machine à états modélisant le système d'alarme.



2) Écrire le programme permettant d'implémenter ce système d'alarme.

```
#include <unistd.h>
#include <pigpio.h>
#include <thread>
#include <atomic>

constexpr uint8_t B1=2; // B1 connecté à BCM2
constexpr uint8_t B2=3; // B1 connecté à BCM3
constexpr uint8_t ALARM=21; // diode LED connecté à BCM21

std::atomic<bool> enFlash=false;

void flasher(){
	while (1) {
		while (enFlash) {
			gpioWrite(ALARM,!gpioRead(ALARM));
			sleep(1);
		}
		gpioWrite(ALARM,0);
	}
}

int main(int argc, char *argv[]) {
	gpioInitialise();
	gpioSetMode(B1, PI_INPUT);
	gpioSetMode(B2,PI_INPUT);
	gpioSetPullUpDown(B1,PI_PUD_UP);
	gpioSetPullUpDown(B2,PI_PUD_UP);

	gpioSetMode(ALARM,PI_OUTPUT);
	gpioWrite(ALARM,0);
	enum class state {INIT,START,WAIT};
	state etat=state::INIT;
	uint32_t v;
	std::thread myThread(flasher);
	while (1) {
		switch(etat) {
		case state::INIT:
			enFlash=false;
			if (gpioRead(B1)==0) {
				while (gpioRead(B1)==0);
				etat=state::START;
			}
			break;
		case state::START:
			enFlash=true;
			v=gpioTick();
			etat=state::WAIT;
			break;
		case state::WAIT:
			enFlash=true;
			if (gpioRead(B1)==0) {
				while (gpioRead(B1)==0);
				etat=state::START;
			}
			else if (gpioRead(B2)==0 || gpioTick()-v>10000000) {
				while (gpioRead(B2)==0);
```

```
                        etat=state::INIT;
                }
                break;
        }
    }
    gpioTerminate();
    return 0;
}
```

3) Reprendre le même programme en utilisant les interruptions pour la la vérification de l'état des boutons poussoirs.

```
#include <unistd.h>
#include <pigpio.h>

constexpr uint8_t B1=2; // B1 connecté à BCM2
constexpr uint8_t B2=3; // B1 connecté à BCM3
constexpr uint8_t ALARM=21; // diode LED connecté à BCM21
volatile bool B1_evt=false,B2_evt=false;
void ISR(int gpio, int level, uint32_t   tick) {
    if (gpio==B1) B1_evt=true;
    else if (gpio==B2) B2_evt=true;
}

std::atomic<bool> enFlash=false;

void flasher(){
    while (1) {
        while (enFlash) {
            gpioWrite(ALARM,!gpioRead(ALARM));
            sleep(1);
        }
        gpioWrite(ALARM,0);
    }
}



int main(int argc, char *argv[]) {
    gpioInitialise();
    gpioSetMode(B1, PI_INPUT);
    gpioSetMode(B2,PI_INPUT);
    gpioSetPullUpDown(B1,PI_PUD_UP);
    gpioSetPullUpDown(B2,PI_PUD_UP);
    gpioSetISRFunc(B1,FALLING_EDGE,-1,ISR);
    gpioSetISRFunc(B2,FALLING_EDGE,-1,ISR);
    gpioSetMode(ALARM,PI_OUTPUT);
    gpioWrite(ALARM,0);
    enum class state {INIT,START,WAIT};
    state etat=state::INIT;
    uint32_t v;
    while (1) {
        switch(etat) {
        case state::INIT:
            enFlash=false;
            if (B1_evt) {
                etat=state::START;
```

```
                }
                B1_evt=B2_evt=false;
                break;
            case state::START:
                enFlash=true;
                v=gpioTick();
                etat=state::WAIT;
                B1_evt=B2_evt=false;
                break;
            case state::WAIT:
                enFlash=true;
                if (B1_evt) {
                    etat=state::START;
                }
                else if (B2_evt || gpioTick()-v>10000000) {
                    etat=state::INIT;
                }
                B1_evt=B2_evt=false;
                break;
            }
        }
        gpioTerminate();
        return 0;
}
```

Exercice 2.

Dans un laboratoire de recherche, une expérience nécessite le maintient d'une salle à une température fixe. Le refroidissement ne doit être en aucun cas interrompu.
Pour ce faire, on utilise deux bouteilles de gaz réfrigérant B1 et B2 utilisées en alternance. Dès qu'une bouteille est vide, le système doit commuter immédiatement sur l'autre et vice-versa. Cela laisse ainsi le temps de changer la bouteille vide pendant que l'autre est utilisée. Si par mégarde, les deux bouteilles venaient à être vides, une alarme se déclenche, afin d'alerter tout le monde sur l'urgence de la situation.

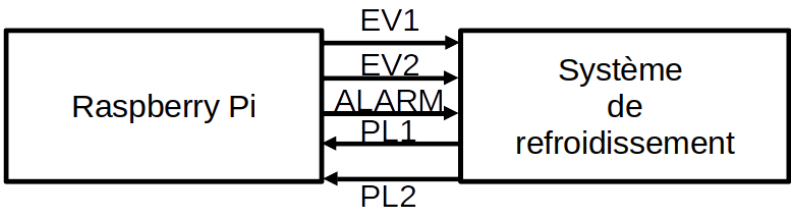
Ce système de surveillance est réalisé à l'aide d'un module admettant les entrées/sorties suivantes :

• PL1,PL2 : Informations issues de capteurs de pression positionnées sur les bouteilles. Ces signaux passent à '1' si la bouteille est vide
• EV1,EV2 : Actionneurs reliés aux électrovannes des bouteilles.
La bouteille i est ouverte et délivre son gaz si `EVi = '1'`
• ALARM : Commande de l'alarme du système, active au niveau haut.

Les contraintes du cahier des charges sont les suivantes :
• À l'état initial, les deux bouteilles sont fermées. On cherchera à utiliser prioritairement la bouteille 1.
• Une bouteille de gaz est utilisée jusqu'à ce qu'elle soit vide. À ce moment uniquement, on commute sur l'autre bouteille.

• L’alarme ne peut être désactivée que par un reset asynchrone du système.
• Après le déclenchement de l’alarme, le système doit reprendre le refroidissement, sans désactiver l’alarme, dès qu’une bouteille pleine soit disponible.

1) Donner la représentation en bloc de fonction du module.



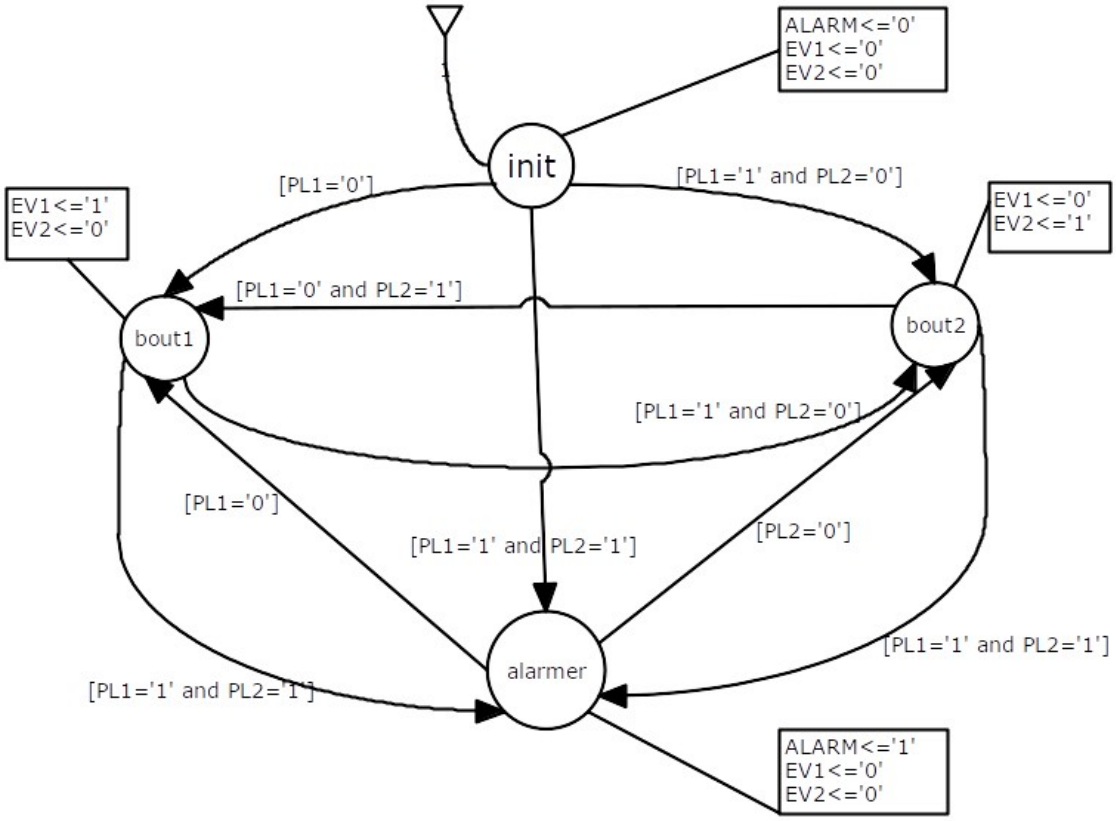
2) Donner le code C++ de la fonction `void initialize()` qui permet de configurer les entrées/sorties.

On propose les connexions suivantes : EV1 BCM14, EV2 BCM15, ALARM BCM16, PL1 BCM17, PL2 BCM18

```
constexpr uint8_t EV1 = 14;
constexpr uint8_t EV2 = 15;
constexpr uint8_t ALARM = 16;
constexpr uint8_t PL1 = 17;
constexpr uint8_t PL2 = 18;

void initialize() {
      gpioSetMode(EV1, PI_OUTPUT);
      gpioWrite(EV1,0);
      gpioSetMode(EV2, PI_OUTPUT);
      gpioWrite(EV2,0);
      gpioSetMode(ALARM, PI_OUTPUT);
      gpioWrite(ALARM, 0);
      gpioSetMode(PL1, PI_INPUT);
      gpioSetMode(PL2, PI_INPUT);
}
```

3) Donner la machine à états décrivant le comportement du module.



4) Traduire la machine à états en code C++.

```
int main() {
	gpioInitialise();
	initialize();

	enum class state {
		init, bout1, bout2, alarmer
	};
	state etat = state::init;

	while (1) {
		switch (etat) {
		case state::init:
			gpioWrite(EV1, 0);
			gpioWrite(EV2, 0);
			gpioWrite(ALARM, 0);
			if (gpioRead(PL1) == 0) {
				etat = state::bout1;
			} else if (gpioRead(PL1) == 1 && gpioRead(PL2) == 0) {
				etat = state::bout2;
			} else if (gpioRead(PL1) == 1 && gpioRead(PL2) == 1) {
```

```
                        etat = state::alarmer;
                }
                break;
        case state::bout1:
                gpioWrite(EV1, 1);
                gpioWrite(EV2, 0);

                if (gpioRead(PL1) == 1 && gpioRead(PL2) == 0) {
                        etat = state::bout2;
                } else if (gpioRead(PL1) == 1 && gpioRead(PL2) == 1) {
                        etat = state::alarmer;
                }

                break;
        case state::bout2:
                gpioWrite(EV1, 0);
                gpioWrite(EV2, 1);
                if (gpioRead(PL1) == 0 && gpioRead(PL2) == 1) {
                        etat = state::bout1;
                } else if (gpioRead(PL1) == 1 && gpioRead(PL2) == 1) {
                        etat = state::alarmer;
                }

                break;
        case state::alarmer:
                gpioWrite(EV1, 0);
                gpioWrite(EV2, 0);
                gpioWrite(ALARM, 1);
                if (gpioRead(PL1) == 0) {
                        etat = state::bout1;
                } else if (gpioRead(PL2) == 0) {
                        etat = state::bout2;
                }
                break;
        }
    }
    gpioTerminate();
    return 0;
}
```